TOPIC 3 : DIVIDE AND CONQUER

1. **Write a Program to find both the maximum and minimum values in the array. Implement using any programming language of your choice. Execute your code and provide the maximum and minimum values found.**
   Input : N= 8, a[] = {5,7,3,4,9,12,6,2}
   Output : Min = 2, Max = 12
   Test Cases :
   Input : N= 9, a[] = {1,3,5,7,9,11,13,15,17}
   Output : Min = 1, Max = 17
   Test Cases :
   Input : N= 10, a[] = {22,34,35,36,43,67, 12,13,15,17}
   Output : Min 12, Max 67

Program:

```
def find_min_and_max(arr):

    min_val = arr[0]

    max_val = arr[0]



    for num in arr:

        if num < min_val:

            min_val = num

        if num > max_val:

            max_val = num



    return min_val, max_val




# Example 1

array1 = [5, 7, 3, 4, 9, 12, 6, 2]

min_val1, max_val1 = find_min_and_max(array1)

print(f"Input: {array1}")
```

```
print(f"Output: Min = {min_val1}, Max = {max_val1}")


# Example 2

array2 = [1, 3, 5, 7, 9, 11, 13, 15, 17]

min_val2, max_val2 = find_min_and_max(array2)

print(f"Input: {array2}")

print(f"Output: Min = {min_val2}, Max = {max_val2}")


# Example 3

array3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]

min_val3, max_val3 = find_min_and_max(array3)

print(f"Input: {array3}")

print(f"Output: Min = {min_val3}, Max = {max_val3}")
```

2.Consider an array of integers sorted in ascending order: 2,4,6,8,10,12,14,18. Write a Program to find both the maximum and minimum values in the array. Implement using any programming language of your choice. Execute your code and provide the maximum and minimum values found.

> **Input : N=8, 2,4,6,8,10,12,14,18.**
> **Output : Min = 2, Max =18**
> **Test Cases :**
> **Input : N= 9, a[] = {11,13,15,17,19,21,23,35,37}**
> **Output : Min = 11, Max = 37**
> **Test Cases :**
> **Input : N= 10, a[] = {22,34,35,36,43,67, 12,13,15,17}**
> **Output : Min 12, Max 67**

Program:

```
def find_min_and_max(arr):
    min_val = arr[0]
    max_val = arr[-1]

    return min_val, max_val

# Test the function with the given examples

# Example 1
array1 = [2, 4, 6, 8, 10, 12, 14, 18]
min_val1, max_val1 = find_min_and_max(array1)
print(f"Input: {array1}")
```

```
print(f"Output: Min = {min_val1}, Max = {max_val1}")

# Example 2
array2 = [11, 13, 15, 17, 19, 21, 23, 35, 37]
min_val2, max_val2 = find_min_and_max(array2)
print(f"Input: {array2}")
print(f"Output: Min = {min_val2}, Max = {max_val2}")

# Example 3
array3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]
min_val3, max_val3 = find_min_and_max(array3)
print(f"Input: {array3}")
print(f"Output: Min = {min_val3}, Max = {max_val3}")
```

3.You are given an unsorted array 31,23,35,27,11,21,15,28. Write a program for Merge Sort and implement using any programming language of your choice.

Test Cases :
Input : N= 8, a[] = {31,23,35,27,11,21,15,28}
Output : 11,15,21,23,27,28,31,35
Test Cases :
Input : N= 10, a[] = {22,34,25,36,43,67, 52,13,65,17}
Output : 13,17,22,25,34,36,43,52,65,67

Program:

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2

        left_half = arr[:mid]
        right_half = arr[mid:]

        merge_sort(left_half)

        merge_sort(right_half)

        i = j = k = 0

        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
```

```
            j += 1
        k += 1


# Example 1
array1 = [31, 23, 35, 27, 11, 21, 15, 28]
merge_sort(array1)
print(f"Input: [31, 23, 35, 27, 11, 21, 15, 28]")
print(f"Output: {array1}")

# Example 2
array2 = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]
merge_sort(array2)
print(f"Input: [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]")
print(f"Output: {array2}")
```

4.Implement the Merge Sort algorithm in a programming language of your choice and test it on the array 12,4,78,23,45,67,89,1. Modify your implementation to count the number of comparisons made during the sorting process. Print this count along with the sorted array.

Test Cases :
Input : N= 8, a[] = {12,4,78,23,45,67,89,1}
Output : 1,4,12,23,45,67,78,89
Test Cases :
Input : N= 7, a[] = {38,27,43,3,9,82,10}
Output : 3,9,10,27,38,43,82,

Program:

```
def merge_sort(arr):
    global comparison_count
    if len(arr) > 1:
        mid = len(arr) // 2

        left_half = arr[:mid]
        right_half = arr[mid:]

        merge_sort(left_half)

        merge_sort(right_half)

        i = j = k = 0

        while i < len(left_half) and j < len(right_half):
            comparison_count += 1
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
```

```
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1


# Example 1
array1 = [12, 4, 78, 23, 45, 67, 89, 1]
comparison_count = 0
merge_sort(array1)
print(f"Input: [12, 4, 78, 23, 45, 67, 89, 1]")
print(f"Output: {array1}")
print(f"Number of comparisons: {comparison_count}")

# Example 2
array2 = [38, 27, 43, 3, 9, 82, 10]
comparison_count = 0
merge_sort(array2)
print(f"Input: [38, 27, 43, 3, 9, 82, 10]")
print(f"Output: {array2}")
print(f"Number of comparisons: {comparison_count}")
```

5.Given an unsorted array 10,16,8,12,15,6,3,9,5 Write a program to perform Quick Sort. Choose the first element as the pivot and partition the array accordingly. Show the array after this partition. Recursively apply Quick Sort on the sub-arrays formed. Display the array after each recursive call until the entire array is sorted.

Input : N= 9, a[]= {10,16,8,12,15,6,3,9,5}
Output : 3,5,6,8,9,10,12,15,16
Test Cases :
Input : N= 8, a[] = {12,4,78,23,45,67,89,1}
Output : 1,4,12,23,45,67,78,89
Test Cases :
Input : N= 7, a[] = {38,27,43,3,9,82,10}
Output : 3,9,10,27,38,43,82,

Program:

```
def quick_sort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)

        print(f"Array after partitioning with pivot {arr[pi]}: {arr}")

        quick_sort(arr, low, pi - 1)
        quick_sort(arr, pi + 1, high)

def partition(arr, low, high):
    pivot = arr[low]
    left = low + 1
    right = high
```

```
    done = False
    while not done:
        while left <= right and arr[left] <= pivot:
            left = left + 1
        while arr[right] >= pivot and right >= left:
            right = right - 1
        if right < left:
            done = True
        else:
            arr[left], arr[right] = arr[right], arr[left]

    arr[low], arr[right] = arr[right], arr[low]
    return right


# Example 1
array1 = [10, 16, 8, 12, 15, 6, 3, 9, 5]
print(f"Input: {array1}")
quick_sort(array1, 0, len(array1) - 1)
print(f"Output: {array1}")

# Example 2
array2 = [12, 4, 78, 23, 45, 67, 89, 1]
print(f"\nInput: {array2}")
quick_sort(array2, 0, len(array2) - 1)
print(f"Output: {array2}")

# Example 3
array3 = [38, 27, 43, 3, 9, 82, 10]
print(f"\nInput: {array3}")
quick_sort(array3, 0, len(array3) - 1)
print(f"Output: {array3}")
```

6.Implement the Quick Sort algorithm in a programming language of your choice and test it on the array 19,72,35,46,58,91,22,31. Choose the middle element as the pivot and partition the array accordingly. Show the array after this partition. Recursively apply Quick Sort on the sub-arrays formed. Display the array after each recursive call until the entire array is sorted. Execute your code and show the sorted array.

Input : N= 8, a[] = {19,72,35,46,58,91,22,31}
Output : 19,22,31,35,46,58,72,91
Test Cases :
Input : N= 8, a[] = {31,23,35,27,11,21,15,28}
Output : 11,15,21,23,27,28,31,35
Test Cases :
Input : N= 10, a[] = {22,34,25,36,43,67, 52,13,65,17}
Output : 13,17,22,25,34,36,43,52,65,67

Program:

```
def quick_sort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
```

```
        print(f"Array after partitioning with pivot {arr[pi]}: {arr}")

        quick_sort(arr, low, pi - 1)
        quick_sort(arr, pi + 1, high)

def partition(arr, low, high):
    mid = (low + high) // 2
    pivot = arr[mid]

    arr[mid], arr[high] = arr[high], arr[mid]
    pivot_index = high

    i = low - 1

    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[pivot_index] = arr[pivot_index], arr[i + 1]
    return i + 1


# Example 1
array1 = [19, 72, 35, 46, 58, 91, 22, 31]
print(f"Input: {array1}")
quick_sort(array1, 0, len(array1) - 1)
print(f"Output: {array1}")

# Example 2
array2 = [31, 23, 35, 27, 11, 21, 15, 28]
print(f"\nInput: {array2}")
quick_sort(array2, 0, len(array2) - 1)
print(f"Output: {array2}")

# Example 3
array3 = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]
print(f"\nInput: {array3}")
quick_sort(array3, 0, len(array3) - 1)
print(f"Output: {array3}")
```

7.Implement the Binary Search algorithm in a programming language of your choice and test it on the array 5,10,15,20,25,30,35,40,45 to find the position of the element 20. Execute your code and provide the index of the element 20. Modify your implementation to count the number of comparisons made during the search process. Print this count along with the result.

Input : N= 9, a[] = {5,10,15,20,25,30,35,40,45}, search key = 20
Output : 4
Test cases
Input : N= 6, a[] = {10,20,30,40,50,60}, search key = 50
Output : 5
Input : N= 7, a[] = {21,32,40,54,65,76,87}, search key = 32
Output : 2

Program:

```
def binary_search(arr, key):
    low = 0
    high = len(arr) - 1
    comparisons = 0

    while low <= high:
        mid = (low + high) // 2
        comparisons += 1

        if arr[mid] == key:
            return mid, comparisons
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1

    return -1, comparisons

# Test the function with the given example

# Example 1
array1 = [5, 10, 15, 20, 25, 30, 35, 40, 45]
search_key1 = 20
index1, comparisons1 = binary_search(array1, search_key1)
print(f"Input: {array1}, search key: {search_key1}")
print(f"Output: Index = {index1}, Comparisons = {comparisons1}")

# Example 2
array2 = [10, 20, 30, 40, 50, 60]
search_key2 = 50
index2, comparisons2 = binary_search(array2, search_key2)
print(f"\nInput: {array2}, search key: {search_key2}")
print(f"Output: Index = {index2}, Comparisons = {comparisons2}")

# Example 3
array3 = [21, 32, 40, 54, 65, 76, 87]
search_key3 = 32
index3, comparisons3 = binary_search(array3, search_key3)
print(f"\nInput: {array3}, search key: {search_key3}")
print(f"Output: Index = {index3}, Comparisons = {comparisons3}")
```

8.You are given a sorted array 3,9,14,19,25,31,42,47,53 and asked to find the position of the element 31 using Binary Search. Show the mid-point calculations and the steps involved in finding the element. Display, what would happen if the array was not sorted, how would this impact the performance and correctness of the Binary Search algorithm?

Input : N= 9, a[] = {3,9,14,19,25,31,42,47,53}, search key = 31
Output : 6
Test cases
Input : N= 7, a[] = {13,19,24,29,35,41,42}, search key = 42
Output : 7
Test cases
Input : N= 6, a[] = {20,40,60,80,100,120}, search key = 60

Output : 3

Program:

```
def binary_search(arr, key):
    low = 0
    high = len(arr) - 1
    comparisons = 0

    while low <= high:
        mid = (low + high) // 2
        comparisons += 1
        print(f"Step {comparisons}: low = {low}, high = {high}, mid = {mid}, arr[mid] = {arr[mid]}")

        if arr[mid] == key:
            return mid, comparisons
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1

    return -1, comparisons

# Test the function with the given example

# Example 1
array1 = [3, 9, 14, 19, 25, 31, 42, 47, 53]
search_key1 = 31
print(f"Input: {array1}, search key: {search_key1}")
index1, comparisons1 = binary_search(array1, search_key1)
print(f"Output: Index = {index1}, Comparisons = {comparisons1}")

# Example 2
array2 = [13, 19, 24, 29, 35, 41, 42]
search_key2 = 42
print(f"\nInput: {array2}, search key: {search_key2}")
index2, comparisons2 = binary_search(array2, search_key2)
print(f"Output: Index = {index2}, Comparisons = {comparisons2}")

# Example 3
array3 = [20, 40, 60, 80, 100, 120]
search_key3 = 60
print(f"\nInput: {array3}, search key: {search_key3}")
index3, comparisons3 = binary_search(array3, search_key3)
print(f"Output: Index = {index3}, Comparisons = {comparisons3}")
```

9.Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).

(i) Input : points = [[1,3],[-2,2],[5,8],[0,1]],k=2
Output:[[-2, 2], [0, 1]]

(ii) Input: points = [[1, 3], [-2, 2]], k = 1
Output: [[-2, 2]]

(iii) Input: points = [[3, 3], [5, -1], [-2, 4]], k = 2

Output: [[3, 3], [-2, 4]]

Program:

```
import heapq

def k_closest_points(points, k):
    max_heap = []

    for x, y in points:
        dist = -(x*x + y*y)

        heapq.heappush(max_heap, (dist, [x, y]))

        if len(max_heap) > k:
            heapq.heappop(max_heap)

    return [point for dist, point in max_heap]

# Test cases
points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k1 = 2
print(f"Input: points = {points1}, k = {k1}")
print("Output:", k_closest_points(points1, k1))

points2 = [[1, 3], [-2, 2]]
k2 = 1
print(f"\nInput: points = {points2}, k = {k2}")
print("Output:", k_closest_points(points2, k2))

points3 = [[3, 3], [5, -1], [-2, 4]]
k3 = 2
print(f"\nInput: points = {points3}, k = {k3}")
print("Output:", k_closest_points(points3, k3))
```

10.Given four lists A, B, C, D of integer values,Write a program to compute how many tuples n(i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.

(i) Input: A = [1, 2], B = [-2, -1], C = [-1, 2], D = [0, 2]

Output: 2

(ii) Input: A = [0], B = [0], C = [0], D = [0]

Output: 1

Program:

```
from collections import defaultdict

def four_sum_count(A, B, C, D):
    sum_ab = defaultdict(int)

    for a in A:
```

```
        for b in B:
            sum_ab[a + b] += 1

    count = 0

    for c in C:
        for d in D:
            target = -(c + d)
            if target in sum_ab:
                count += sum_ab[target]

    return count

# Test cases
A1 = [1, 2]
B1 = [-2, -1]
C1 = [-1, 2]
D1 = [0, 2]
print(f"Input: A = {A1}, B = {B1}, C = {C1}, D = {D1}")
print("Output:", four_sum_count(A1, B1, C1, D1))

A2 = [0]
B2 = [0]
C2 = [0]
D2 = [0]
print(f"\nInput: A = {A2}, B = {B2}, C = {C2}, D = {D2}")
print("Output:", four_sum_count(A2, B2, C2, D2))
```

11.To Implement the Median of Medians algorithm ensures that you handle the worst-case time complexity efficiently while finding the k-th smallest element in an unsorted array.

arr = [12, 3, 5, 7, 19] k = 2 Expected Output:5
arr = [12, 3, 5, 7, 4, 19, 26] k = 3 Expected Output:5
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6 Expected Output:6

program:

```
import statistics

def find_kth_smallest(arr, k):
    def partition(arr, left, right):
        if right - left < 5:
            return sorted(arr[left:right+1])

        chunks = [arr[i:i+5] for i in range(left, right+1, 5)]
        medians = [statistics.median(chunk) for chunk in chunks]

        return medians
```

```
    def median_of_medians(arr, left, right, k):
        if left == right:
            return arr[left]

        medians = partition(arr, left, right)
        pivot = median_of_medians(medians, 0, len(medians)-1, len(medians)//2)

        lower = [x for x in arr if x < pivot]
        equal = [x for x in arr if x == pivot]
        upper = [x for x in arr if x > pivot]

        if k < len(lower):
            return median_of_medians(lower, 0, len(lower)-1, k)
        elif k < len(lower) + len(equal):
            return pivot
        else:
            return median_of_medians(upper, 0, len(upper)-1, k - len(lower) - len(equal))

    if k < 1 or k > len(arr):
        raise ValueError("k is out of bounds")

    return median_of_medians(arr, 0, len(arr)-1, k-1)

# Test cases
arr1 = [12, 3, 5, 7, 19]
k1 = 2
print(f"Input: arr = {arr1}, k = {k1}")
print("Output:", find_kth_smallest(arr1, k1))

arr2 = [12, 3, 5, 7, 4, 19, 26]
k2 = 3
print(f"\nInput: arr = {arr2}, k = {k2}")
print("Output:", find_kth_smallest(arr2, k2))

arr3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k3 = 6
print(f"\nInput: arr = {arr3}, k = {k3}")
print("Output:", find_kth_smallest(arr3, k3))
```

12.To Implement a function median_of_medians(arr, k) that takes an unsorted array arr and an integer k, and returns the k-th smallest element in the array.

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6
arr = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27] k = 5
Output: An integer representing the k-th smallest element in the array.

Program:

```
import statistics
```

```
def median_of_medians(arr, k):
    def partition(arr, left, right):
        if right - left < 5:
            return sorted(arr[left:right+1])

        chunks = [arr[i:i+5] for i in range(left, right+1, 5)]
        medians = [statistics.median(chunk) for chunk in chunks]

        return medians

    def select(arr, left, right, k):
        if left == right:
            return arr[left]

        medians = partition(arr, left, right)
        pivot = select(medians, 0, len(medians)-1, len(medians)//2)

        lower = [x for x in arr if x < pivot]
        equal = [x for x in arr if x == pivot]
        upper = [x for x in arr if x > pivot]

        if k < len(lower):
            return select(lower, 0, len(lower)-1, k)
        elif k < len(lower) + len(equal):
            return pivot
        else:
            return select(upper, 0, len(upper)-1, k - len(lower) - len(equal))

    if k < 1 or k > len(arr):
        raise ValueError("k is out of bounds")

    return select(arr, 0, len(arr)-1, k-1)

# Test cases
arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k1 = 6
print(f"Input: arr = {arr1}, k = {k1}")
print("Output:", median_of_medians(arr1, k1))

arr2 = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27]
k2 = 5
print(f"\nInput: arr = {arr2}, k = {k2}")
print("Output:", median_of_medians(arr2, k2))
```

13.Write a program to implement Meet in the Middle Technique. Given an array of integers and a target sum, find the subset whose sum is closest to the target. You will use the Meet in the Middle technique to efficiently find this subset.

a) Set[] = {45, 34, 4, 12, 5, 2}　　　　Target Sum : 42

b) Set[]= {1, 3, 2, 7, 4, 6}   Target sum = 10:

program:

```
import sys

def meet_in_the_middle(arr, target_sum):
    n = len(arr)
    half = n // 2

    sums1 = []
    for i in range(1 << half):
        subset_sum = 0
        for j in range(half):
            if i & (1 << j):
                subset_sum += arr[j]
        sums1.append(subset_sum)

    sums2 = []
    for i in range(1 << (n - half)):
        subset_sum = 0
        for j in range(n - half):
            if i & (1 << j):
                subset_sum += arr[half + j]
        sums2.append(subset_sum)

    sums2.sort()

    closest_sum = sys.maxsize
    closest_diff = sys.maxsize

    for sum1 in sums1:
        required_sum = target_sum - sum1
        lo, hi = 0, len(sums2) - 1
        while lo <= hi:
            mid = (lo + hi) // 2
            current_sum = sums2[mid]
            if current_sum == required_sum:
                return target_sum
            elif current_sum < required_sum:
                lo = mid + 1
            else:
                hi = mid - 1
            if abs(current_sum - required_sum) < closest_diff:
                closest_diff = abs(current_sum - required_sum)
                closest_sum = current_sum

    return closest_sum

# Test cases
```

```
set1 = [45, 34, 4, 12, 5, 2]
target_sum1 = 42
print(f"Input: Set = {set1}, Target Sum = {target_sum1}")
print("Closest Subset Sum:", meet_in_the_middle(set1, target_sum1))

set2 = [1, 3, 2, 7, 4, 6]
target_sum2 = 10
print(f"\nInput: Set = {set2}, Target Sum = {target_sum2}")
print("Closest Subset Sum:", meet_in_the_middle(set2, target_sum2))
```

14.Write a program to implement Meet in the Middle Technique. Given a large array of integers and an exact sum E, determine if there is any subset that sums exactly to E. Utilize the Meet in the Middle technique to handle the potentially large size of the array. Return true if there is a subset that sums exactly to E, otherwise return false.

a) E = {1, 3, 9, 2, 7, 12} exact Sum = 15

b) E = {3, 34, 4, 12, 5, 2} exact Sum = 15

program:

```
def meet_in_the_middle_subset_sum(arr, exact_sum):
    n = len(arr)
    half = n // 2

    def generate_subset_sums(arr, start, end):
        subset_sums = set()
        for i in range(1 << (end - start)):
            subset_sum = 0
            for j in range(end - start):
                if i & (1 << j):
                    subset_sum += arr[start + j]
            subset_sums.add(subset_sum)
        return subset_sums

    sums1 = generate_subset_sums(arr, 0, half)

    sums2 = generate_subset_sums(arr, half, n)

    for sum1 in sums1:
        if exact_sum - sum1 in sums2:
            return True

    return False

# Test cases
E1 = [1, 3, 9, 2, 7, 12]
exact_sum1 = 15
print(f"Input: E = {E1}, Exact Sum = {exact_sum1}")
print("Subset with exact sum exists:", meet_in_the_middle_subset_sum(E1, exact_sum1))
```

```
E2 = [3, 34, 4, 12, 5, 2]
exact_sum2 = 15
print(f"\nInput: E = {E2}, Exact Sum = {exact_sum2}")
print("Subset with exact sum exists:", meet_in_the_middle_subset_sum(E2, exact_sum2))
```

15.Given two 2×2 Matrices A and B

A=(1 7 B=(1 3

3 5) 7 5)

Use Strassen's matrix multiplication algorithm to compute the product matrix C such that C=A×B.

Test Cases:

Consider the following matrices for testing your implementation:

Test Case 1:

A=(1 7 B=(6 8
3 5), 4 2)

Expected Output:
C=(18 14
62 66)

Program:

```
def strassen_matrix_multiply(A, B):
    a11, a12, a21, a22 = A[0][0], A[0][1], A[1][0], A[1][1]
    b11, b12, b21, b22 = B[0][0], B[0][1], B[1][0], B[1][1]

    M1 = (a11 + a22) * (b11 + b22)
    M2 = (a21 + a22) * b11
    M3 = a11 * (b12 - b22)
    M4 = a22 * (b21 - b11)
    M5 = (a11 + a12) * b22
    M6 = (a21 - a11) * (b11 + b12)
    M7 = (a12 - a22) * (b21 + b22)

    c11 = M1 + M4 - M5 + M7
    c12 = M3 + M5
    c21 = M2 + M4
    c22 = M1 - M2 + M3 + M6

    C = [[c11, c12], [c21, c22]]
```

```
    return C

# Test case 1
A1 = [[1, 7], [3, 5]]
B1 = [[6, 8], [4, 2]]
print("Matrix A:")
for row in A1:
    print(row)
print("\nMatrix B:")
for row in B1:
    print(row)
print("\nComputing A * B using Strassen's algorithm:")
C1 = strassen_matrix_multiply(A1, B1)
print("\nResult Matrix C:")
for row in C1:
    print(row)
```

16. Given two integers X=1234 and Y=5678: Use the Karatsuba algorithm to compute the product Z=X x Y

Test Case 1:

Input: x=1234,y=5678

Expected Output: z=1234×5678=7016652

Program:

```
def karatsuba_multiply(x, y):

    if x < 10 or y < 10:

        return x * y



    str_x = str(x)

    str_y = str(y)

    n = max(len(str_x), len(str_y))



    m = (n + 1) // 2
```

```
    high_x = x // (10 ** m)
    low_x = x % (10 ** m)
    high_y = y // (10 ** m)
    low_y = y % (10 ** m)

    z0 = karatsuba_multiply(low_x, low_y)
    z1 = karatsuba_multiply((low_x + high_x), (low_y + high_y))
    z2 = karatsuba_multiply(high_x, high_y)

    return (z2 * (10 ** (2 * m))) + ((z1 - z2 - z0) * (10 ** m)) + z0

# Test case
X = 1234
Y = 5678
print(f"Input: X = {X}, Y = {Y}")
Z = karatsuba_multiply(X, Y)
print(f"Output: Z = {Z}")
```